PS11: QT Programming

CMPE 230 - Spring 2024

Based on the slides by Abdullatif Köksal, with his permission.

What is QT?

Qt is a free and open-source widget toolkit for creating **graphical user interfaces** as well as **cross-platform** applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems with little or no change in the underlying codebase while still being a native application with native capabilities and speed.

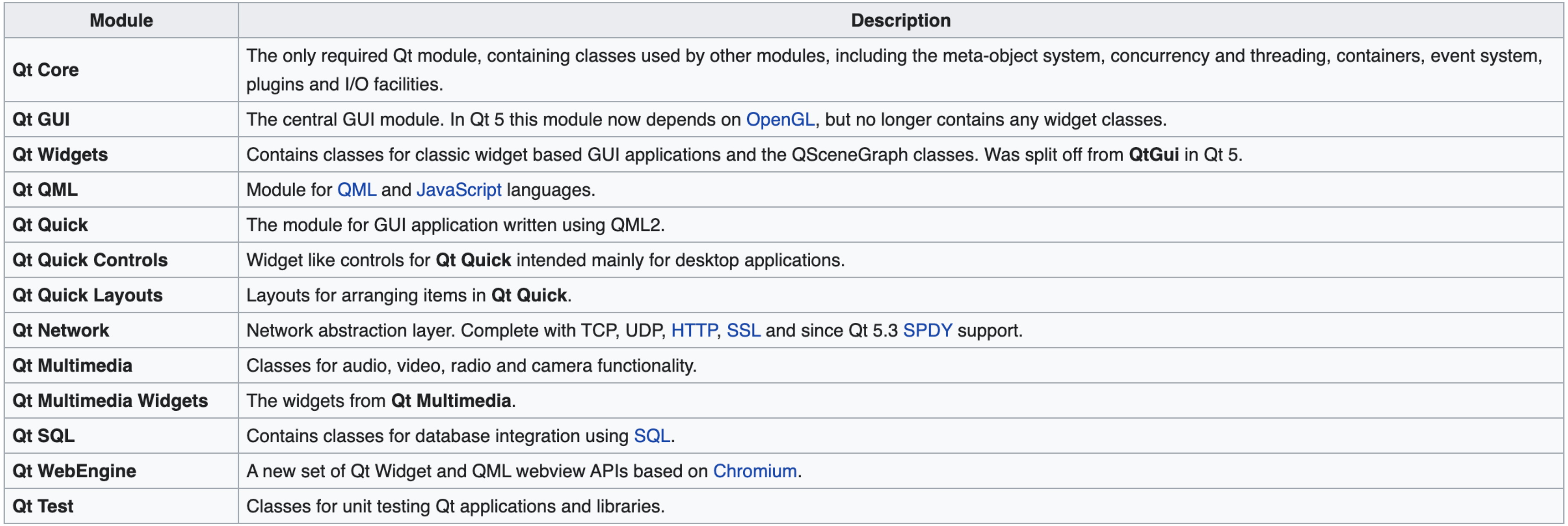
https://en.wikipedia.org/wiki/Qt_(software)

How to install QT on Ubuntu

- QT's website
- Terminal

```
sudo apt-get install build-essential

sudo apt install -y qtcreator qtbase5-dev qt5-qmake cmake
```

QT Modules



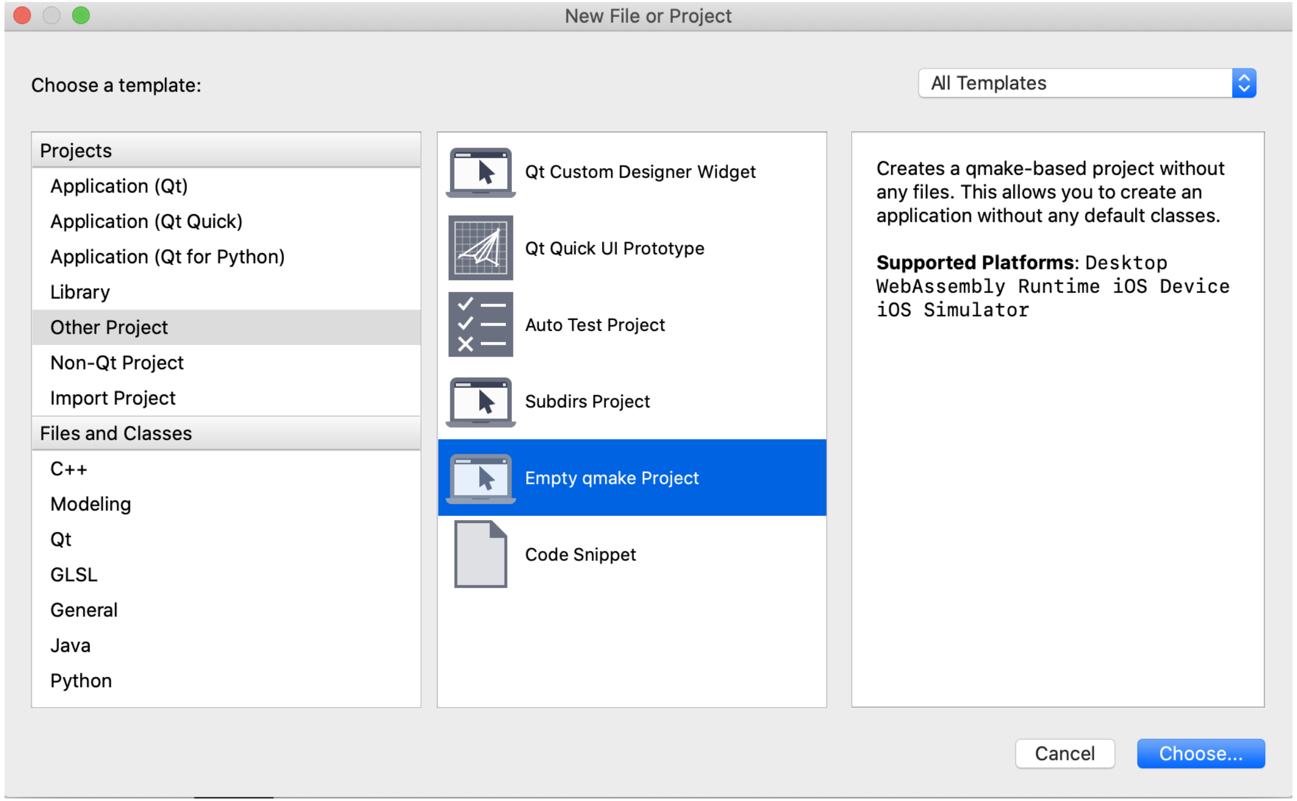
Module	Description
Qt Core	The only required Qt module, containing classes used by other modules, including the meta-object system, concurrency and threading, containers, event system, plugins and I/O facilities.
Qt GUI	The central GUI module. In Qt 5 this module now depends on OpenGL, but no longer contains any widget classes.
Qt Widgets	Contains classes for classic widget based GUI applications and the QSceneGraph classes. Was split off from **QtGui** in Qt 5.
Qt QML	Module for QML and JavaScript languages.
Qt Quick	The module for GUI application written using QML2.
Qt Quick Controls	Widget like controls for **Qt Quick** intended mainly for desktop applications.
Qt Quick Layouts	Layouts for arranging items in **Qt Quick**.
Qt Network	Network abstraction layer. Complete with TCP, UDP, HTTP, SSL and since Qt 5.3 SPDY support.
Qt Multimedia	Classes for audio, video, radio and camera functionality.
Qt Multimedia Widgets	The widgets from **Qt Multimedia**.
Qt SQL	Contains classes for database integration using SQL.
Qt WebEngine	A new set of Qt Widget and QML webview APIs based on Chromium.
Qt Test	Classes for unit testing Qt applications and libraries.

https://en.wikipedia.org/wiki/Qt_(software)

1. Hello World Example

Our objective is to create a simple GUI application with a label and a quit button which are aligned horizontally.

Let’s start with an empty qmake project.


New File or Project

Choose a template:

All Templates

Projects
- Application (Qt)
- Application (Qt Quick)
- Application (Qt for Python)
- Library
- Other Project
- Non-Qt Project
- Import Project

Files and Classes
- C++
- Modeling
- Qt
- GLSL
- General
- Java
- Python

- Qt Custom Designer Widget
- Qt Quick UI Prototype
- Auto Test Project
- Subdirs Project
- Empty qmake Project
- Code Snippet

Creates a qmake-based project without any files. This allows you to create an application without any default classes.

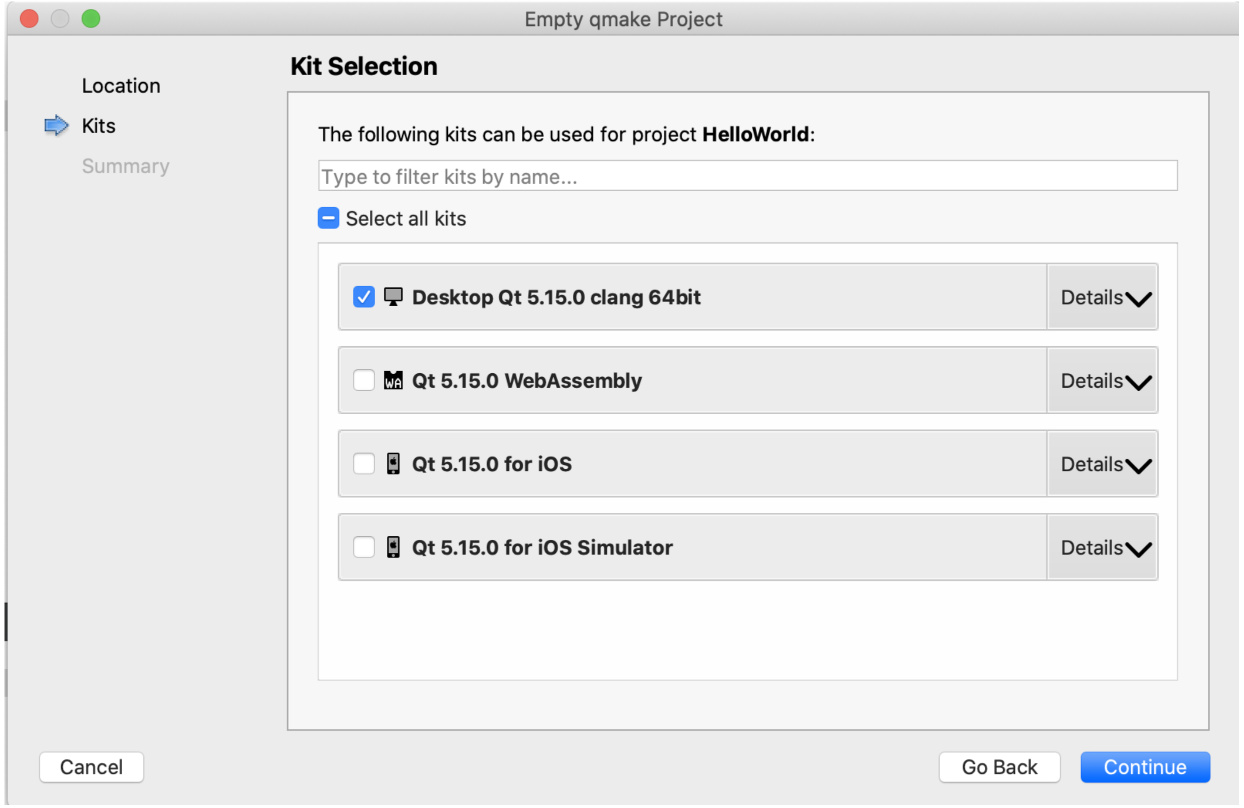
Supported Platforms: `Desktop WebAssembly Runtime iOS Device iOS Simulator`

Cancel | Choose...

1. Hello World Example

Add Desktop kit for desktop applications.

Note that C++ compiler might be different.


Empty qmake Project

Location
Kits
Summary

Kit Selection

The following kits can be used for project **HelloWorld**:

Type to filter kits by name...

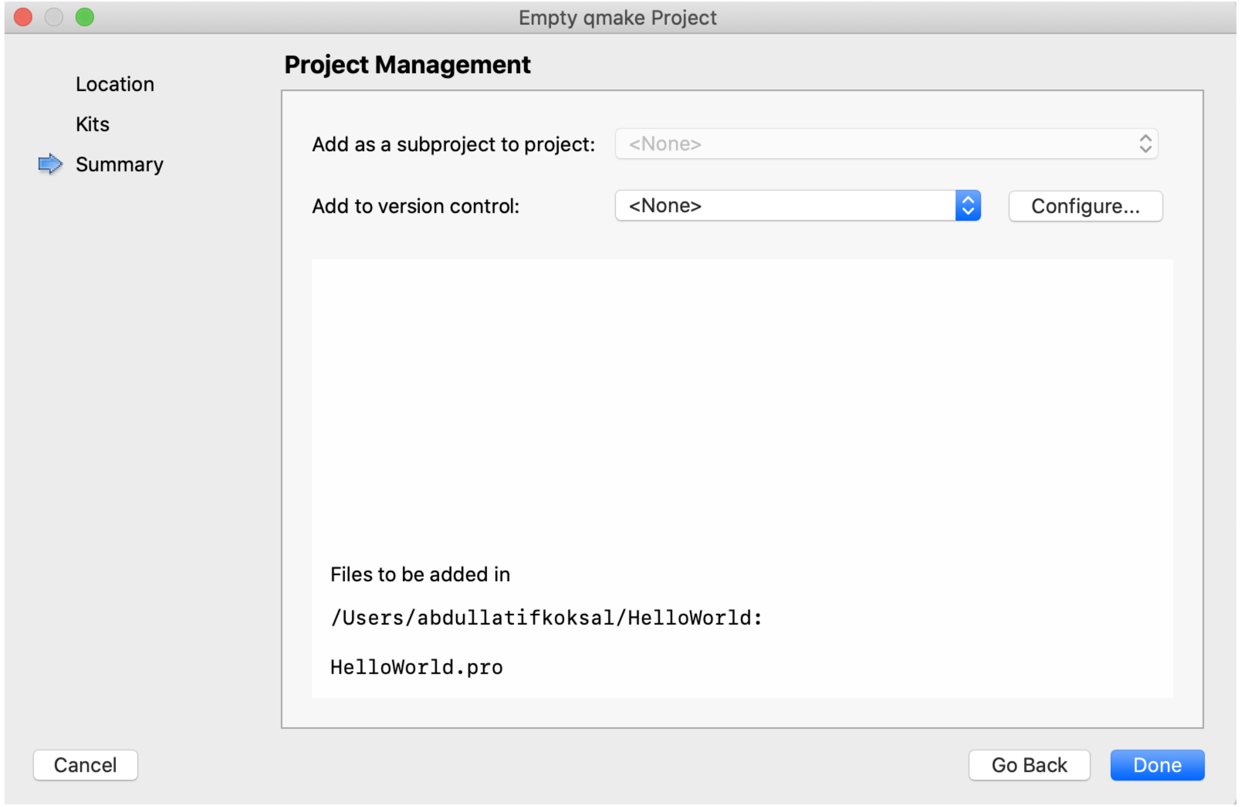
Select all kits

- [✓] **Desktop Qt 5.15.0 clang 64bit** Details
- [ ] **Qt 5.15.0 WebAssembly** Details
- [ ] **Qt 5.15.0 for iOS** Details
- [ ] **Qt 5.15.0 for iOS Simulator** Details

Cancel | Go Back | Continue

1. Hello World Example

It will create our project with only .pro file.



Add as a subproject to project: <None>

Add to version control: <None>

Files to be added in

```
/Users/abdullatifkoksal/HelloWorld:

HelloWorld.pro
```

.pro File

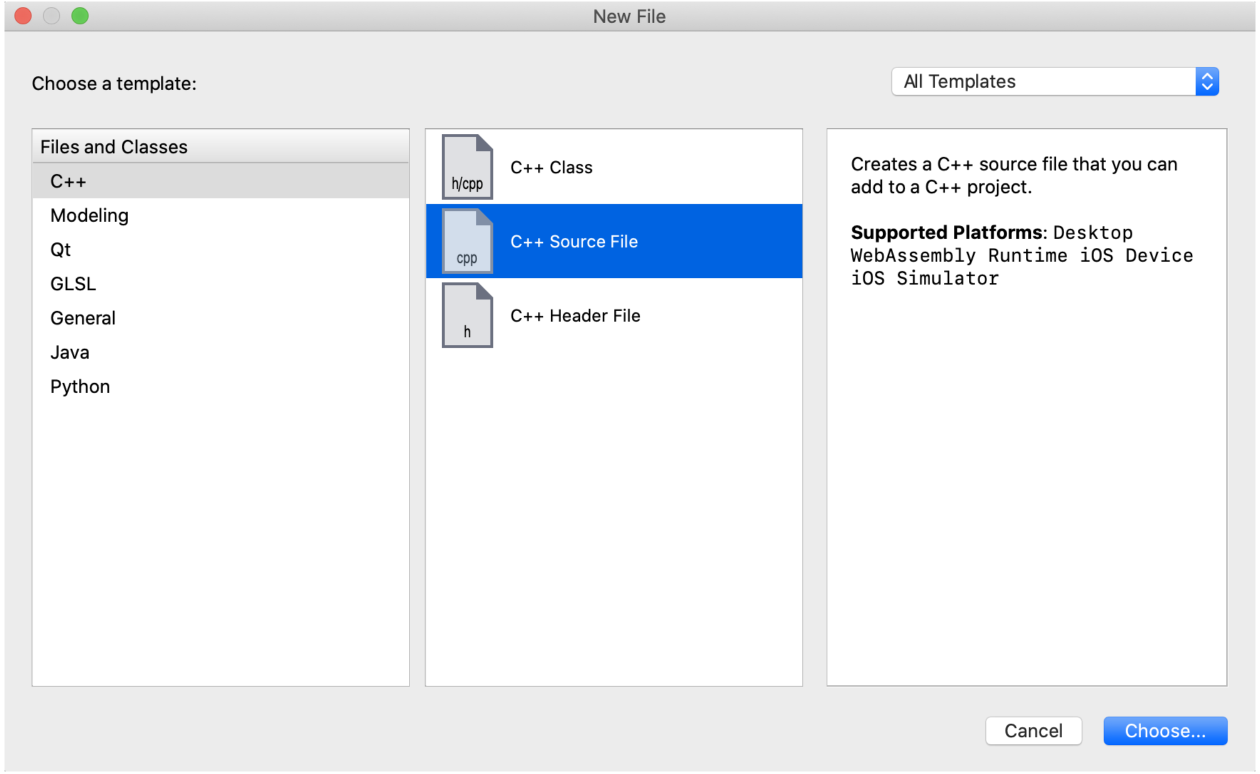
Project files contain all the information **required by qmake** to build your application, library, or plugin. Generally, you use series of declarations to specify the resources in the project, but support for simple programming constructs enables you to describe different build processes for different platforms and environments.

We will use project files to add source and header files (it will be done automatically) and to **declare required QT Libraries.**

https://doc.qt.io/qt-5/qmake-project-files.html

1. Hello World Example

Let's create a C++ file for our main window.

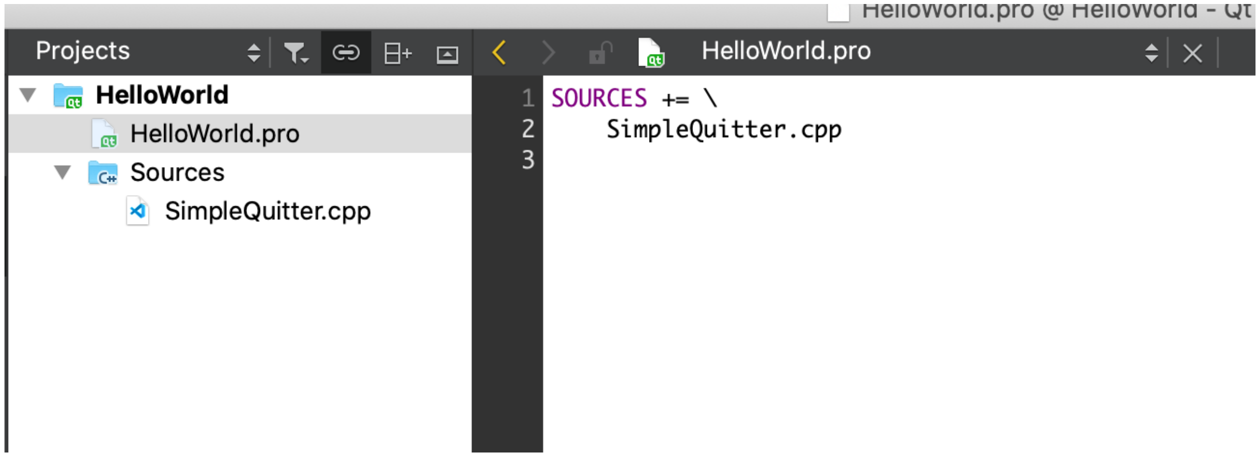


Creates a C++ source file that you can add to a C++ project.

Supported Platforms: Desktop WebAssembly Runtime iOS Device iOS Simulator

1. Hello World Example

It's added to project file automatically.

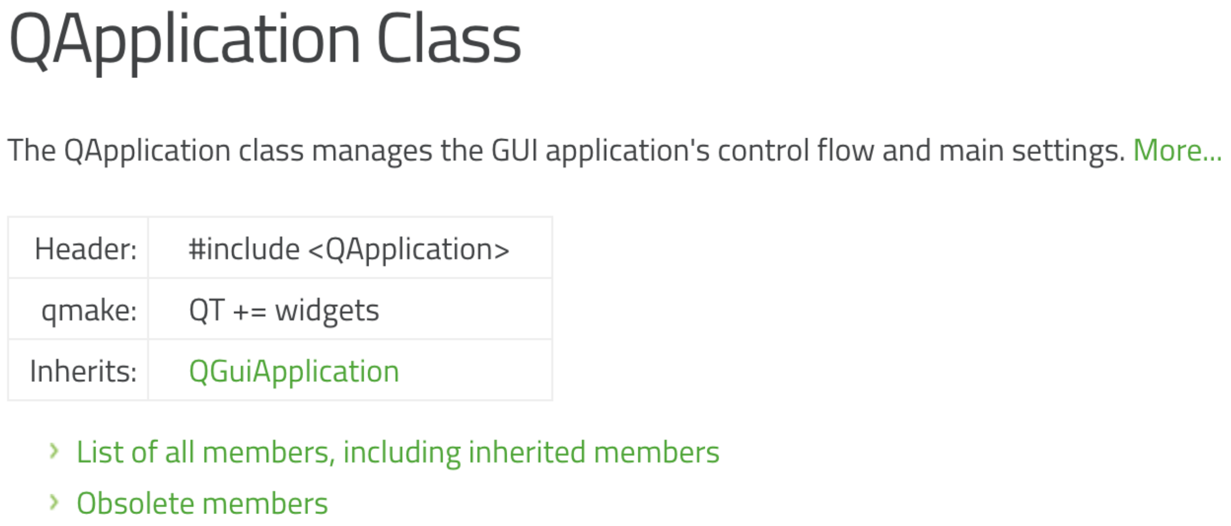


```
SOURCES += \
    SimpleQuitter.cpp

```

QApplication

It handles widget specific initialization, finalization.

For any GUI application using Qt, there is precisely one QApplication object, no matter whether the application has 0, 1, 2 or more windows at any given time.



QApplication Class

The QApplication class manages the GUI application's control flow and main settings. More...

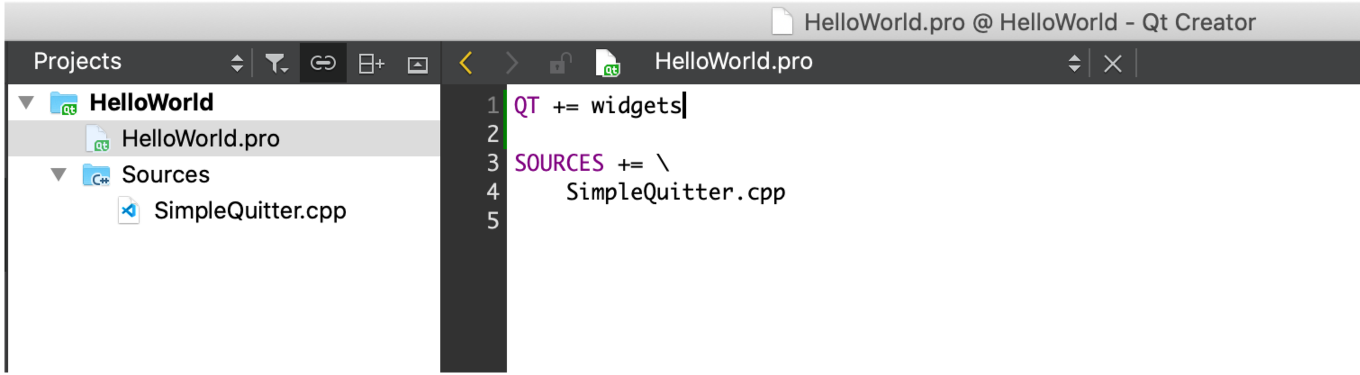
Header:	#include <QApplication>
qmake:	QT += widgets
Inherits:	QGuiApplication

- List of all members, including inherited members
- Obsolete members

https://doc.qt.io/qt-5/qapplication.html

.pro File

We have to add
QT += widgets
to our project file.

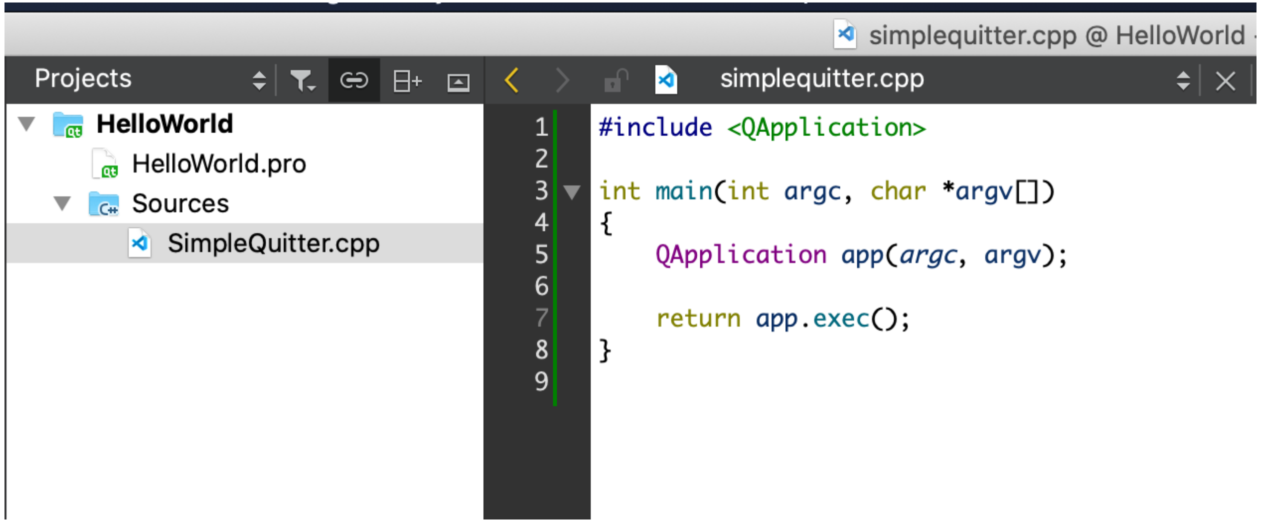


```
QT += widgets

SOURCES += \
    SimpleQuitter.cpp

```

https://doc.qt.io/qt-5/qapplication.html

.cpp File

And our simple main file:



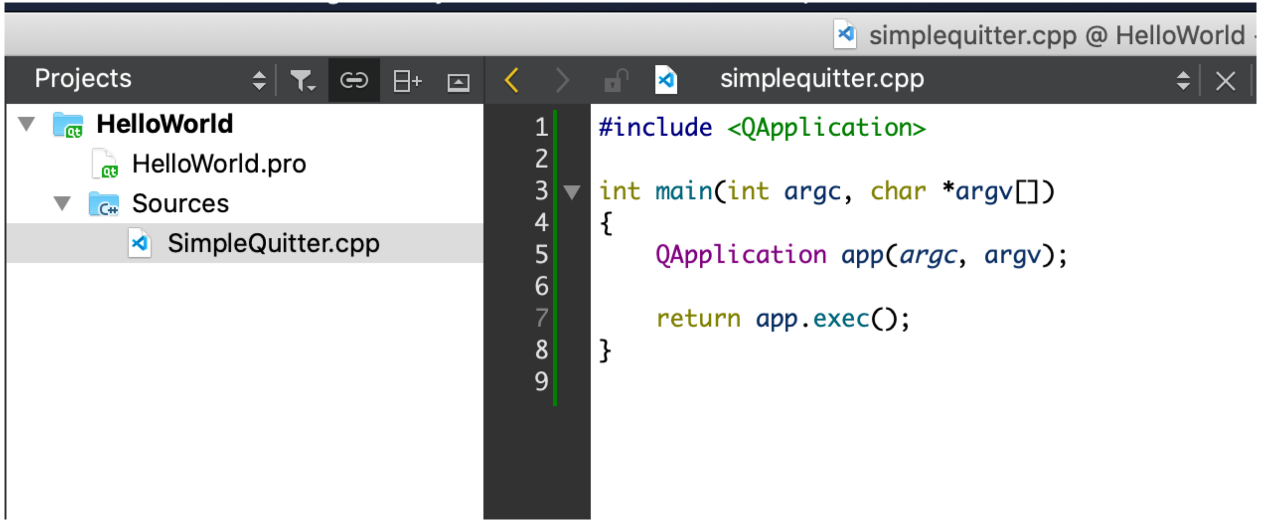
```
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    return app.exec();
}
```

https://doc.qt.io/qt-5/qapplication.html

1. Hello World Example

And our simple main file:

We can run our application but there will be no element in it.

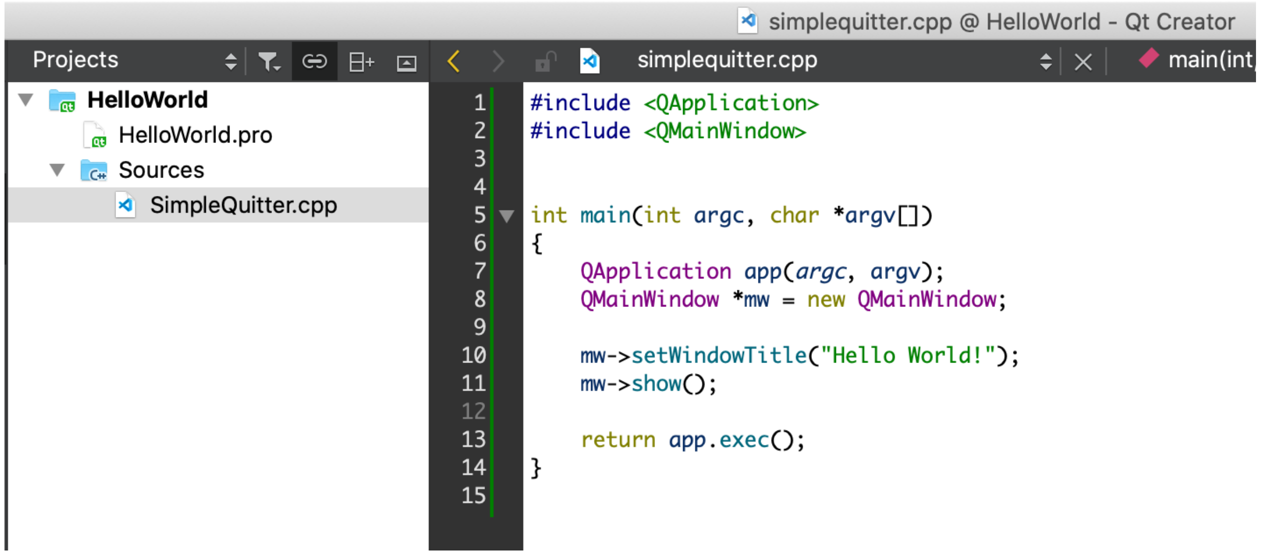


```
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    return app.exec();
}
```

https://doc.qt.io/qt-5/qapplication.html

1. Hello World Example

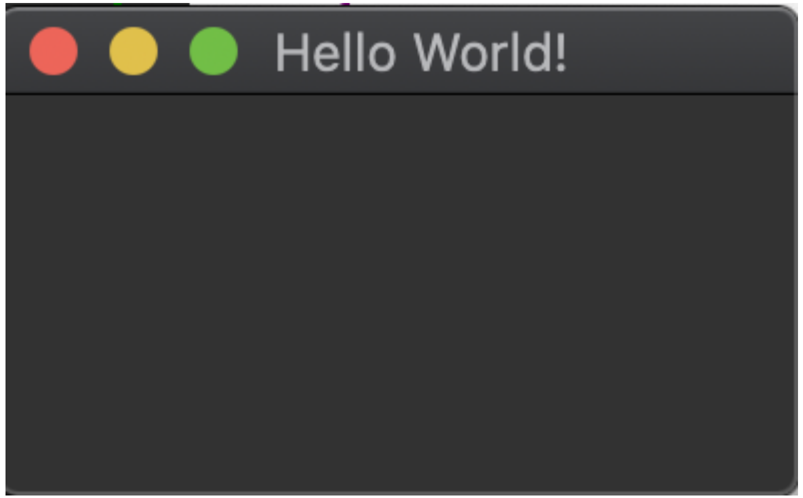
Let's add QMainWindow class for common needs of a main window which include status bar, toolbar, and menu bar.



```cpp
#include <QApplication>
#include <QMainWindow>


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMainWindow *mw = new QMainWindow;

    mw->setWindowTitle("Hello World!");
    mw->show();

    return app.exec();
}

```

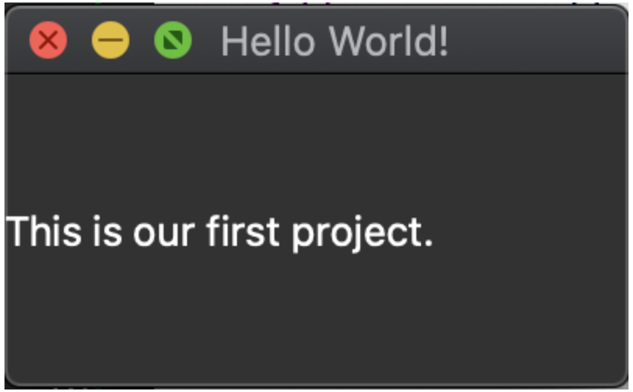
1. Hello World Example

Our program would look like this. It has already OS specific design which has quit, minimize, and full-screen buttons.

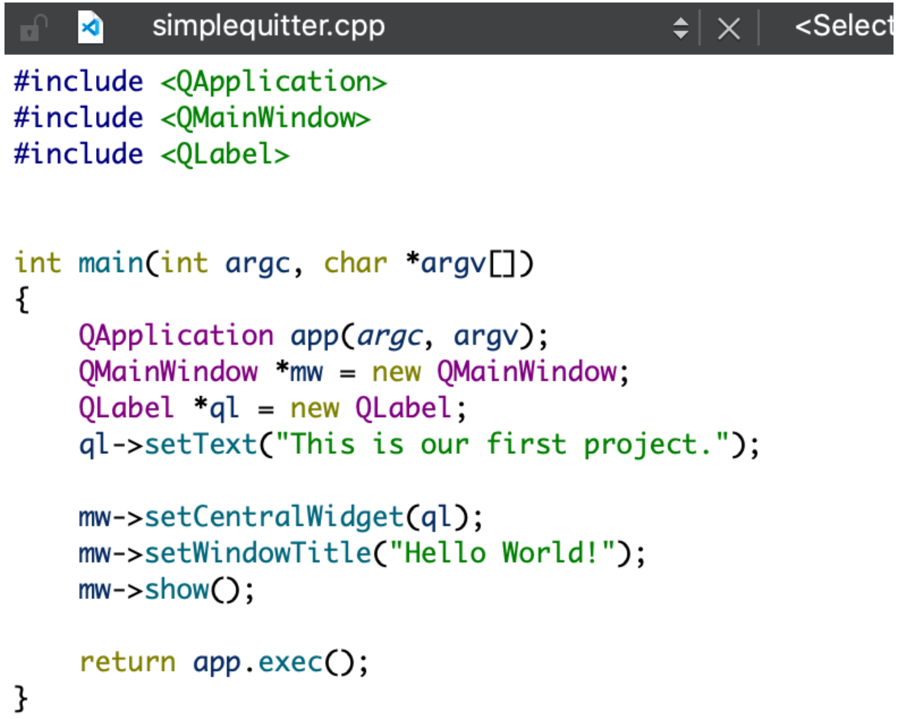


1. Hello World Example

Let's add a label that says this is our first project and add it as our central widget to mainwindow.



This is our first project.



```
#include <QApplication>
#include <QMainWindow>
#include <QLabel>


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMainWindow *mw = new QMainWindow;
    QLabel *ql = new QLabel;
    ql->setText("This is our first project.");

    mw->setCentralWidget(ql);
    mw->setWindowTitle("Hello World!");
    mw->show();

    return app.exec();
}
```

1. Hello World Example

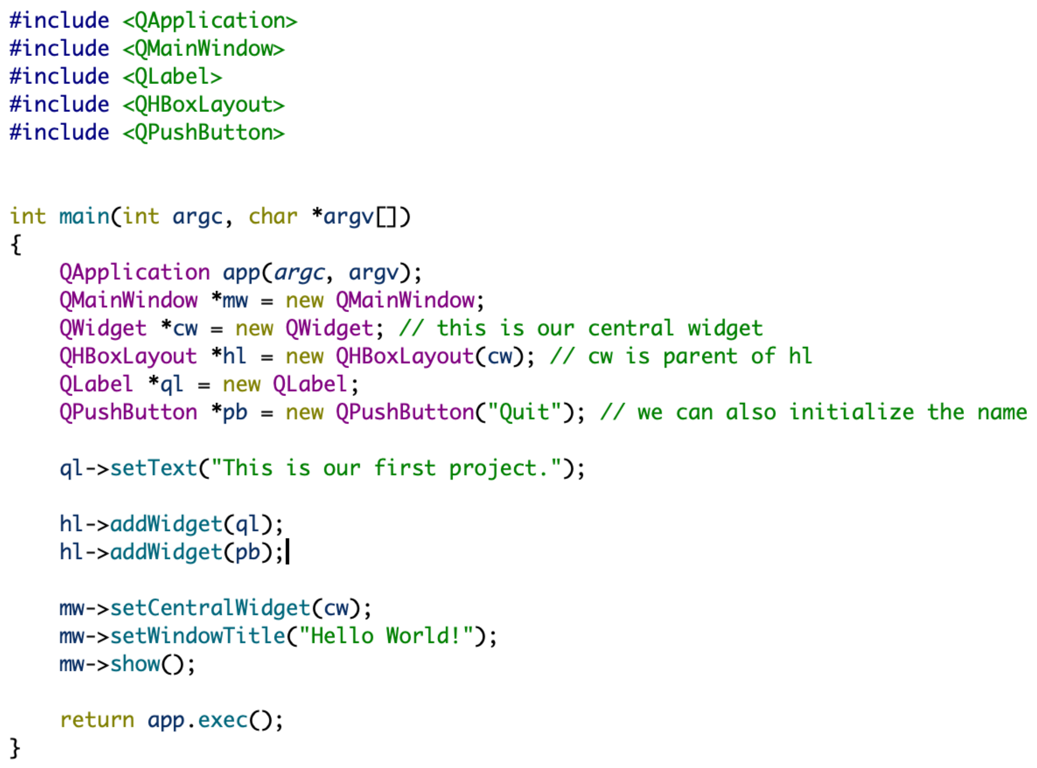
Now we have to add button which will be aligned horizontally with our label. So, we have to add QHBoxLayout first as our central widget.

cw is our central widget and the parent of hl. QLabel is added as a widget to hl.

```cpp
#include <QApplication>
#include <QMainWindow>
#include <QLabel>
#include <QHBoxLayout>


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMainWindow *mw = new QMainWindow;
    QWidget *cw = new QWidget; // this is our central widget
    QHBoxLayout *hl = new QHBoxLayout(cw); // cw is parent of hl
    QLabel *ql = new QLabel;
    ql->setText("This is our first project.");


    hl->addWidget(ql);
    mw->setCentralWidget(cw);
    mw->setWindowTitle("Hello World!");
    mw->show();

    return app.exec();
}
```

1. Hello World Example

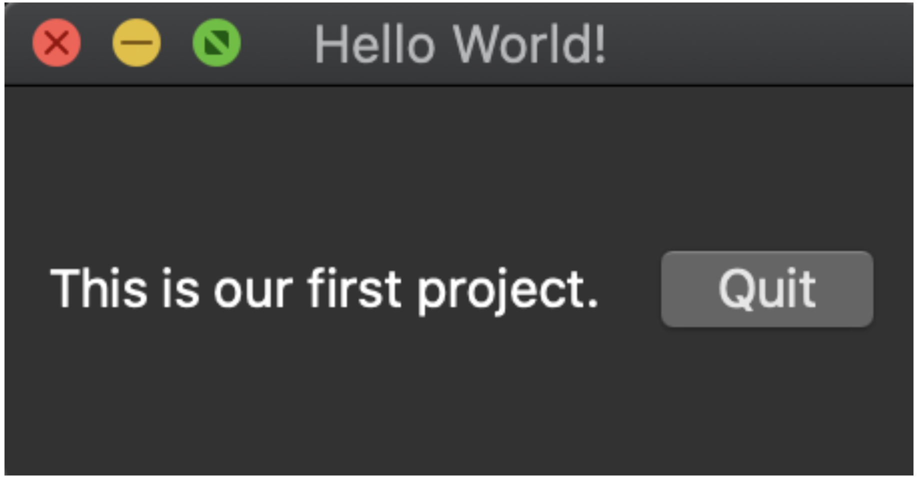
Let’s add the button to QHBoxLayout with its name in initialization.



```
#include <QApplication>
#include <QMainWindow>
#include <QLabel>
#include <QHBoxLayout>
#include <QPushButton>


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMainWindow *mw = new QMainWindow;
    QWidget *cw = new QWidget; // this is our central widget
    QHBoxLayout *hl = new QHBoxLayout(cw); // cw is parent of hl
    QLabel *ql = new QLabel;
    QPushButton *pb = new QPushButton("Quit"); // we can also initialize the name

    ql->setText("This is our first project.");

    hl->addWidget(ql);
    hl->addWidget(pb);

    mw->setCentralWidget(cw);
    mw->setWindowTitle("Hello World!");
    mw->show();

    return app.exec();
}
```

1. Hello World Example

We have a label and a button that are aligned horizontally in the center of main window. However, we didn't add any functionality to button right now.



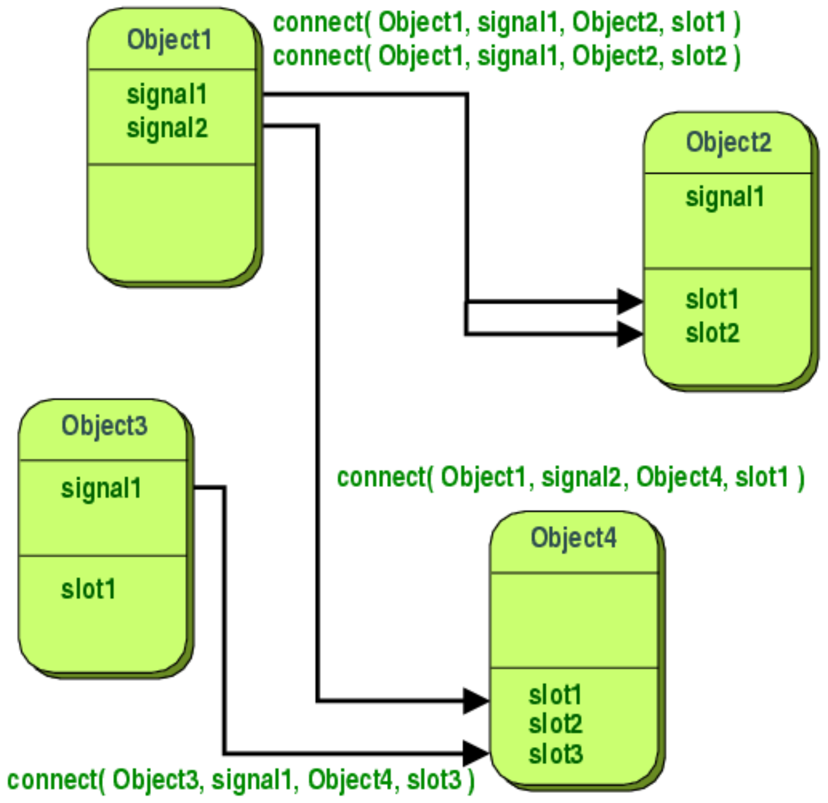
Signals and Slots

Signals:

Signals are **emitted by an object** when its internal state has changed in some way that might be interesting to the object's client or owner.

Slots:

A slot is called when a signal connected to it is emitted. Slots are normal C++ functions and can be called normally; their only special feature is **that signals can be connected to them.**


connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object1
signal1
signal2
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

https://doc.qt.io/qt-5/signalsandslots.html

connect

We can use connect statement to connect signals and slots for wanted action.

```
connect(sender, SIGNAL(signal), receiver, SLOT(slot));
```

1. Hello World Example

Let’s connect our button’s clicked signal to app’s quit slot.

```
#include <QApplication>
#include <QMainWindow>
#include <QLabel>
#include <QHBoxLayout>
#include <QPushButton>


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMainWindow *mw = new QMainWindow;
    QWidget *cw = new QWidget; // this is our central widget
    QHBoxLayout *hl = new QHBoxLayout(cw); // cw is parent of hl
    QLabel *ql = new QLabel;
    QPushButton *pb = new QPushButton("Quit"); // we can also initialize the name

    ql->setText("This is our first project.");

    hl->addWidget(ql);
    hl->addWidget(pb);

    QObject::connect(pb, SIGNAL(clicked()), &app, SLOT(quit()));

    mw->setCentralWidget(cw);
    mw->setWindowTitle("Hello World!");
    mw->show();

    return app.exec();
}
```

2. Color Game

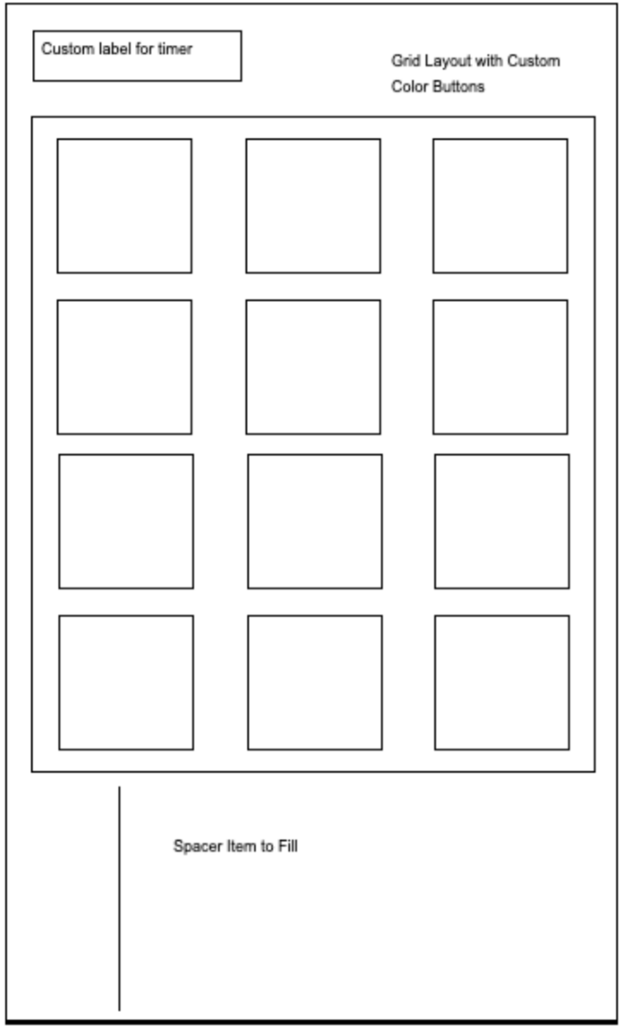
We will build a color game with timer. We would have 4x3 colors, randomly initialized between Blue, Green, and Red. When you click one of them, the color would set again randomly. Your objective is to make all colors same under 30 seconds. We will see:

- QGridLayout
- QVBoxLayout, QHBoxLayout
- QTimer
- QSpacerItem
- QMessageBox
- **Custom Widgets and Layouts**

2. Color Game

Let’s start with some assumptions before coding.

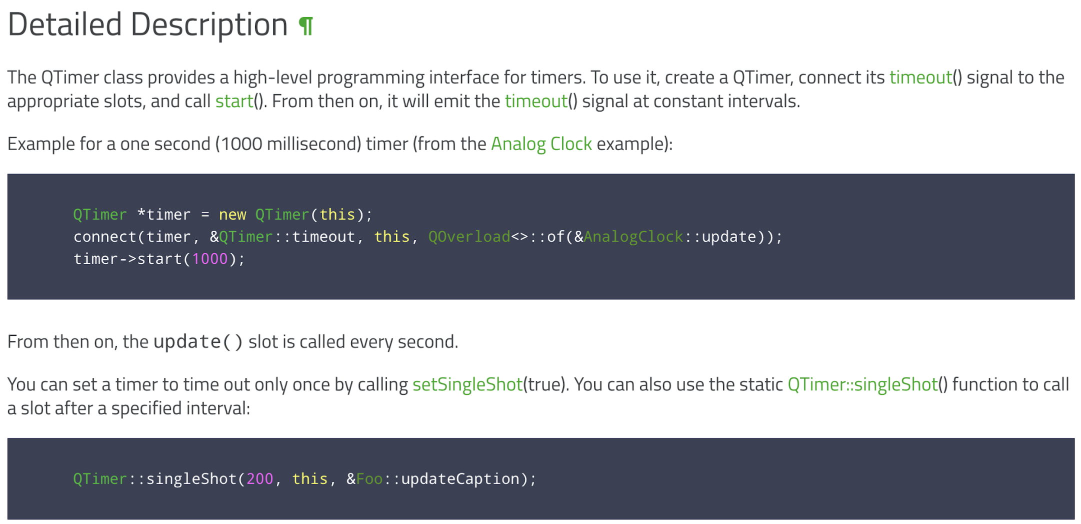
1. We will use a widget as main window instead of QMainWindow.
2. We would have timer label and grid layout in a vertical box layout.
3. We would write custom buttons with colors.
4. We would write custom widget class for timer with labels.
5. We would write custom grid layout which checks colors of all color buttons.



QTimer

Let’s start with the QTimer object.

- Timeout signal will be send based on decided interval (generally with 1 seconds = 1000 milliseconds)
- We will change the label in each signal, so add a slot according to this with a counter.



Detailed Description ¶

The QTimer class provides a high-level programming interface for timers. To use it, create a QTimer, connect its timeout() signal to the appropriate slots, and call start(). From then on, it will emit the timeout() signal at constant intervals.

Example for a one second (1000 millisecond) timer (from the Analog Clock example):

```
    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, QOverload<>::of(&AnalogClock::update));
    timer->start(1000);
```

From then on, the `update()` slot is called every second.

You can set a timer to time out only once by calling setSingleShot(true). You can also use the static QTimer::singleShot() function to call a slot after a specified interval:

```
    QTimer::singleShot(200, this, &Foo::updateCaption);
```

https://doc.qt.io/qt-5/qtimer.html#details

2. Color Game

Add a custom timer class with:

- QTimer to detect seconds
- int counter to count
- QLabel to add it into layout
- MyTimerSlot to change QLabel in each tick.
- We will use message box to declare failure when the timer hits 30.

```
#ifndef MYTIMER_H
#define MYTIMER_H
#include <QTimer>
#include <QLabel>
#include <QMessageBox>

class MyTimer : public QObject
{
    Q_OBJECT

public:
    MyTimer();
    QTimer *timer;
    QLabel *label;
    int counter;

public slots:
    void MyTimerSlot();
};

#endif // MYTIMER_H
```

2. Color Game

Constructor sets label, counter, and timer.

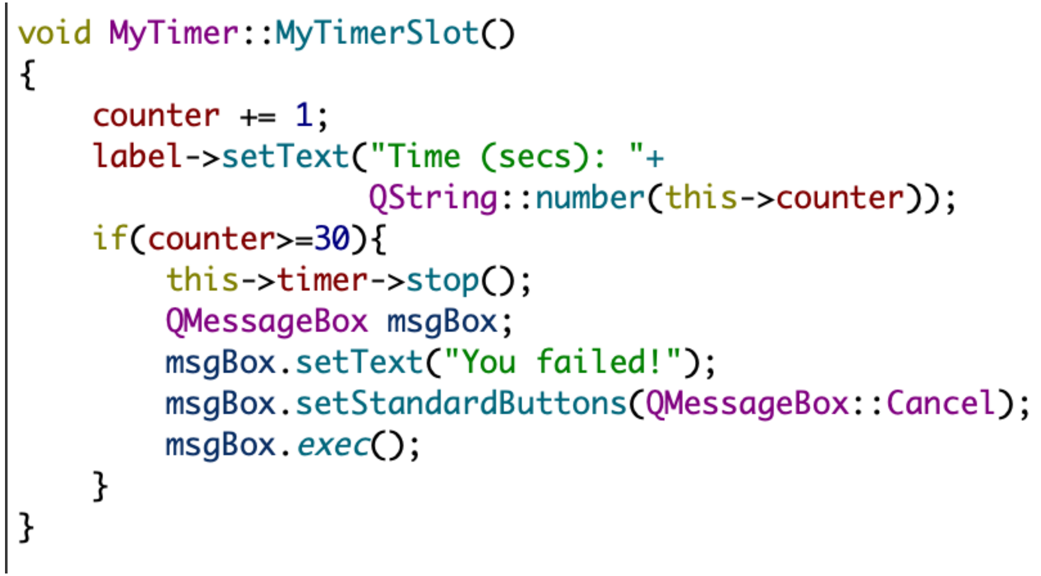
We add a connection MyTimerSlot() for 1000 ms timeout signal.

```
#include "mytimer.h"

MyTimer::MyTimer()
{
    timer = new QTimer(this);
    label = new QLabel("Time (secs): 0");
    counter = 0;

    // setup signal and slot
    connect(timer, SIGNAL(timeout()),
          this, SLOT(MyTimerSlot()));

    timer->start(1000);
}
```

2. Color Game

MyTimerSlot:

- Sets counter in each tick.
- Changes label according to counter.
- If counter is greater than or equal to 30, in other words 30 seconds passed, it raises a message box and stops timer.



```
void MyTimer::MyTimerSlot()
{
    counter += 1;
    label->setText("Time (secs): "+
                       QString::number(this->counter));
    if(counter>=30){
        this->timer->stop();
        QMessageBox msgBox;
        msgBox.setText("You failed!");
        msgBox.setStandardButtons(QMessageBox::Cancel);
        msgBox.exec();
    }
}
```

2. Color Game

Let’s talk about our custom color button which inherits from QPushButton.

- Constructor with color name, text, and parent
- Color name field
- Slot to change color randomly

```
#ifndef COLORBUTTON_H
#define COLORBUTTON_H
#include <QPushButton>
#include <QPalette>


class ColorButton : public QPushButton
{
    Q_OBJECT

public:
    ColorButton(const QString& color,
                const QString& text,
                QWidget* parent = 0);
    QString color;
public slots:
    void change_color();
};

#endif // COLORBUTTON_H
```

2. Color Game

Important notes:

Q_OBJECT statement is necessary to add custom slots.

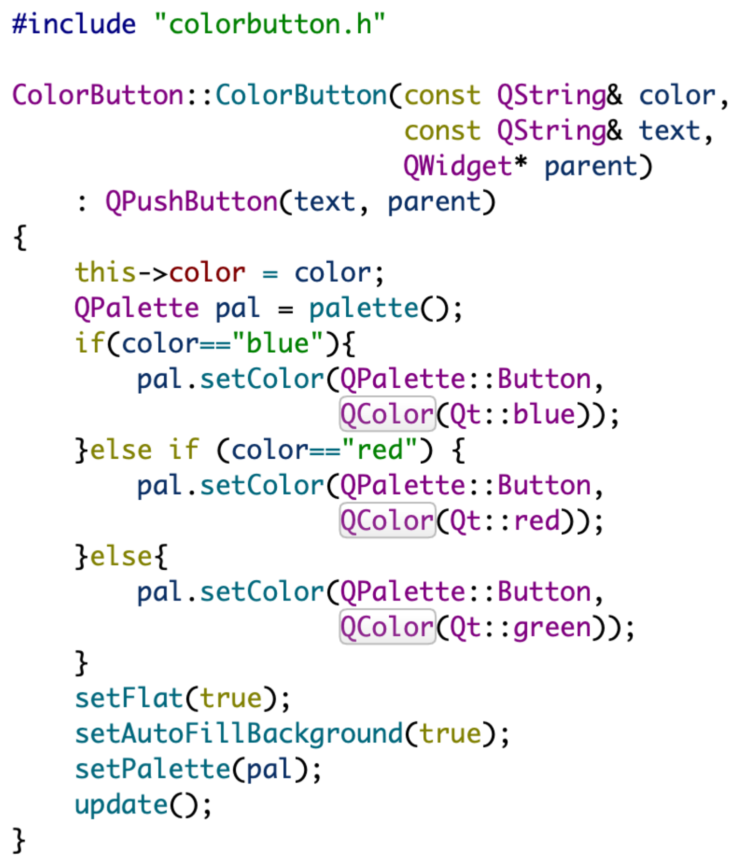
When you add Q_OBJECT, you have to run qmake in your QT Creator to avoid any problems!

```cpp
#ifndef COLORBUTTON_H
#define COLORBUTTON_H
#include <QPushButton>
#include <QPalette>


class ColorButton : public QPushButton
{
    Q_OBJECT

public:
    ColorButton(const QString& color,
                const QString& text,
                QWidget* parent = 0);
    QString color;
public slots:
    void change_color();
};

#endif // COLORBUTTON_H
```

2. Color Game

The constructor:

1. We call constructor method of QPushButton.
2. Set color based on the color string.



```
#include "colorbutton.h"

ColorButton::ColorButton(const QString& color,
                         const QString& text,
                         QWidget* parent)
    : QPushButton(text, parent)
{
    this->color = color;
    QPalette pal = palette();
    if(color=="blue"){
        pal.setColor(QPalette::Button,
                     QColor(Qt::blue));
    }else if (color=="red") {
        pal.setColor(QPalette::Button,
                     QColor(Qt::red));
    }else{
        pal.setColor(QPalette::Button,
                     QColor(Qt::green));
    }
    setFlat(true);
    setAutoFillBackground(true);
    setPalette(pal);
    update();
}
```

2. Color Game

Change color slot:

1. First, create a random integer to detect color.
2. Set color based on this randomization.

P.S.: These functions could have improved with helper functions.

```
void ColorButton::change_color(){
    QPalette pal = palette();
    int color = rand()%3;
    if(color==0){
        this->color = "blue";
        pal.setColor(QPalette::Button,
                            QColor(Qt::blue));
    }else if (color==1) {
        this->color = "red";
        pal.setColor(QPalette::Button,
                            QColor(Qt::red));
    }else{
        this->color = "greed";
        pal.setColor(QPalette::Button,
                            QColor(Qt::green));
    }
    setFlat(true);
    setAutoFillBackground(true);
    setPalette(pal);
    update();
}
```

2. Color Game

1. Now, we have to design QGridLayout. We can put mxn items in Grid Layout. We will put color buttons in 4x3 cells.
2. We have to add a pointer to our timer object to stop it when we match all colors.
3. We also have to check color buttons in this grid layout whether they have same colors or not in each click. So we design it as slots.

```cpp
#ifndef MYGRID_H
#define MYGRID_H
#include <QGridLayout>
#include <QMessageBox>
#include <QTimer>


class MyGrid : public QGridLayout
{
    Q_OBJECT

public:
    MyGrid(QTimer* timer);
    QTimer* timer;

public slots:
    void check_colors();
};


#endif // MYGRID_H
```

2. Color Game

The constructor can be implemented in a straightforward way.

We call constructor of QGridLayout and add a pointer to timer object.

```cpp
#include "mygrid.h"
#include "colorbutton.h"

MyGrid::MyGrid(QTimer *timer) : QGridLayout(){
    this->timer = timer;
}
```

2. Color Game

To check colors, first we implement a very basic logic. Two important things:

1. We check widgets in GridLayout.
2. Then, typecast to ColorButton as we know all widgets are ColorButtons.

Then, we set a message box if all colors are same.

```
void MyGrid::check_colors(){
    bool all_same = true;
    QString prev = "";
    for (int i = 0; i < this->count(); ++i)
    {
      ColorButton *widget = qobject_cast<ColorButton*>
              (this->itemAt(i)->widget());
      if(prev == ""){
          prev = widget->color;
      }else if(prev!=widget->color){
          all_same = false;
      }
    }

    if(all_same){
        this->timer->stop();
        QMessageBox msgBox;
        msgBox.setText("You won!");
        msgBox.exec();
        msgBox.setStandardButtons(QMessageBox::Cancel);
    }
}
```

2. Color Game

Now, we can implement our main window and general logic with the help of these classes.

Our main window is a widget. We defined our custom timer and custom grid layouts.

```
#include <QApplication>
#include <QVBoxLayout>
#include <QSpacerItem>
#include "colorbutton.h"
#include "mytimer.h"
#include "mygrid.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *cw = new QWidget; // this is our main widget
    QVBoxLayout *vb = new QVBoxLayout(cw);
    MyTimer mt;
    MyGrid *gl = new MyGrid(mt.timer);

    for(int row=0; row<4; row++){
        for(int col=0; col<3; col++){
            int color_code = rand()%3;
            QString color;
            if(color_code==0){
                color = "red";
            }else if (color_code==1) {
                color = "blue";
            }else{
                color = "green";
            }
            ColorButton *randButton = new ColorButton(color, "X");
            QObject::connect(randButton, SIGNAL(clicked()),
                             randButton, SLOT(change_color()));
            QObject::connect(randButton, SIGNAL(clicked()),
                             gl, SLOT(check_colors()));
            gl->addWidget(randButton, row, col, 1, 1);
        }
    }
```

2. Color Game

Then, we initialized color buttons with random colors.

We added two connections, both of them is triggered with clicked signal in color button:

- change_color slot in ColorButton is called.
- check_colors slot in GridLayout is called.

```
#include <QApplication>
#include <QVBoxLayout>
#include <QSpacerItem>
#include "colorbutton.h"
#include "mytimer.h"
#include "mygrid.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *cw = new QWidget; // this is our main widget
    QVBoxLayout *vb = new QVBoxLayout(cw);
    MyTimer mt;
    MyGrid *gl = new MyGrid(mt.timer);

    for(int row=0; row<4; row++){
        for(int col=0; col<3; col++){
            int color_code = rand()%3;
            QString color;
            if(color_code==0){
                color = "red";
            }else if (color_code==1) {
                color = "blue";
            }else{
                color = "green";
            }
            ColorButton *randButton = new ColorButton(color, "X");
            QObject::connect(randButton, SIGNAL(clicked()),
                             randButton, SLOT(change_color()));
            QObject::connect(randButton, SIGNAL(clicked()),
                             gl, SLOT(check_colors()));
            gl->addWidget(randButton, row, col, 1, 1);
        }
    }
```

2. Color Game

Finally, we added color button to grid layout based on row and column indices with 1 span.

```cpp
#include <QApplication>
#include <QVBoxLayout>
#include <QSpacerItem>
#include "colorbutton.h"
#include "mytimer.h"
#include "mygrid.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *cw = new QWidget; // this is our main widget
    QVBoxLayout *vb = new QVBoxLayout(cw);
    MyTimer mt;
    MyGrid *gl = new MyGrid(mt.timer);

    for(int row=0; row<4; row++){
        for(int col=0; col<3; col++){
            int color_code = rand()%3;
            QString color;
            if(color_code==0){
                color = "red";
            }else if (color_code==1) {
                color = "blue";
            }else{
                color = "green";
            }
            ColorButton *randButton = new ColorButton(color, "X");
            QObject::connect(randButton, SIGNAL(clicked()),
                             randButton, SLOT(change_color()));
            QObject::connect(randButton, SIGNAL(clicked()),
                             gl, SLOT(check_colors()));
            gl->addWidget(randButton, row, col, 1, 1);
        }
    }
```

2. Color Game

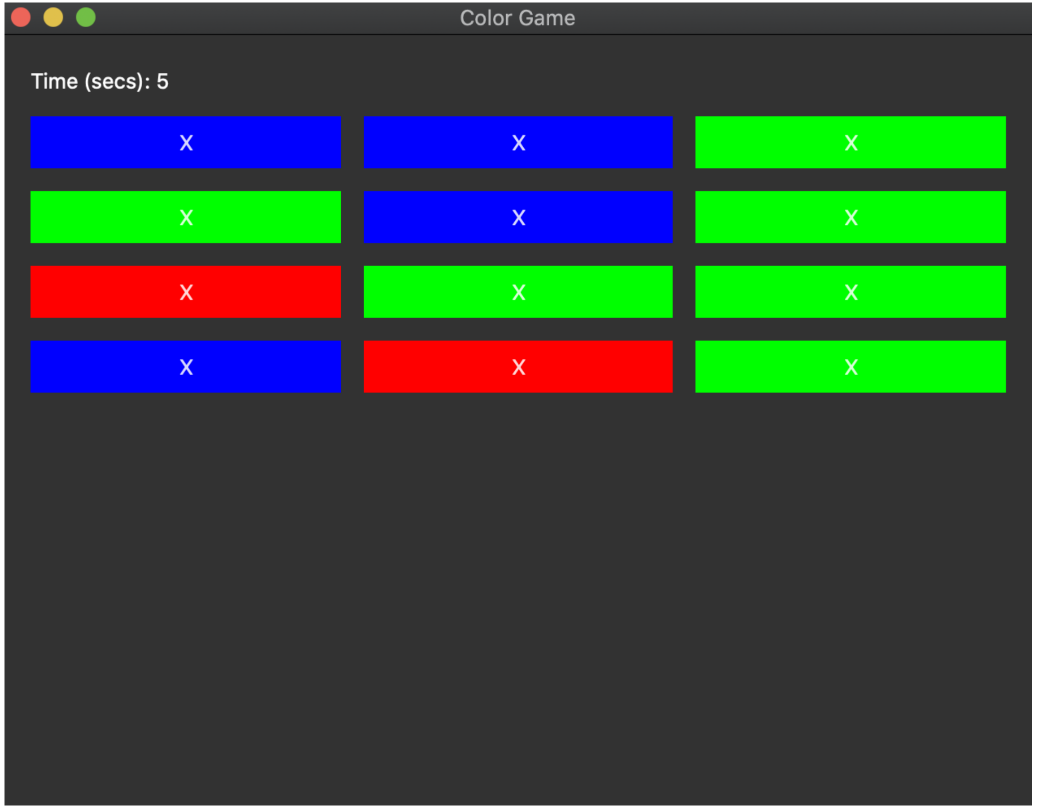
Then, we add widgets and layouts to vertical box with spacer item.

Spacer item expands to add additional space horizontally and vertically.

```cpp
    vb->addWidget(mt.label);
    vb->addLayout(gl);

    QSpacerItem *si = new QSpacerItem(0, 10,
                                      QSizePolicy::Expanding,
                                      QSizePolicy::Expanding);
    vb->addSpacerItem(si);


    cw->setWindowTitle("Color Game");
    cw->resize(640, 480);
    cw->show();

    return app.exec();
}
```

2. Color Game

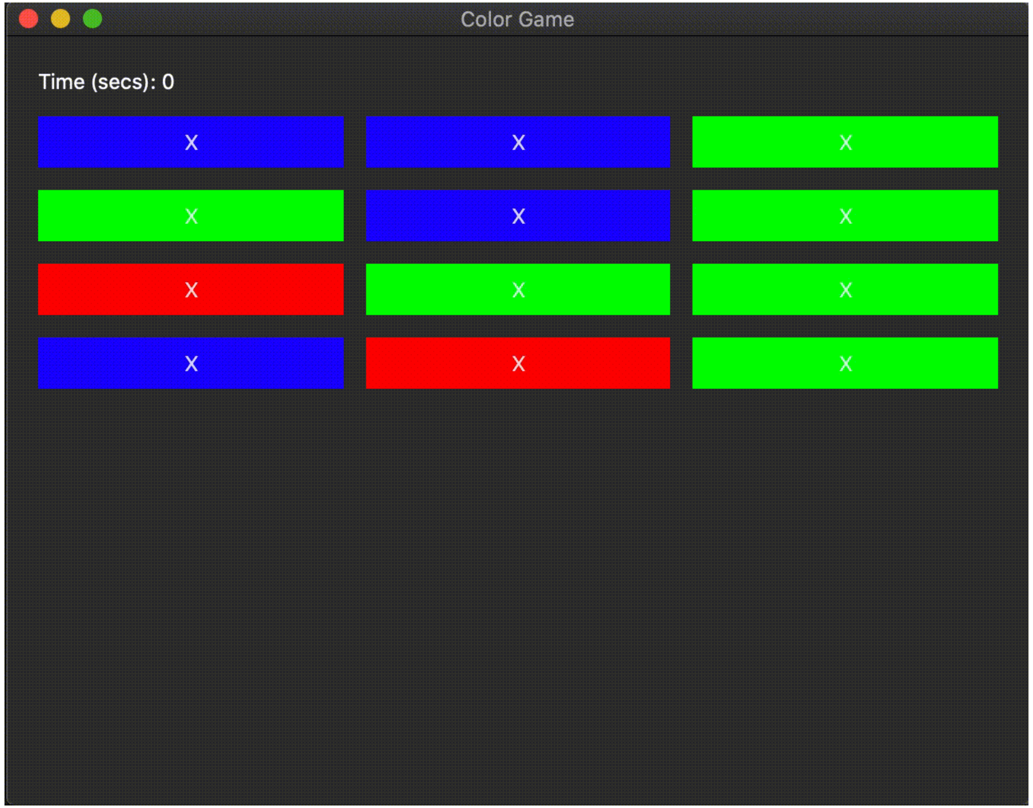
Finally, the design looks like this:



2. Color Game

Finally, the design looks like this:

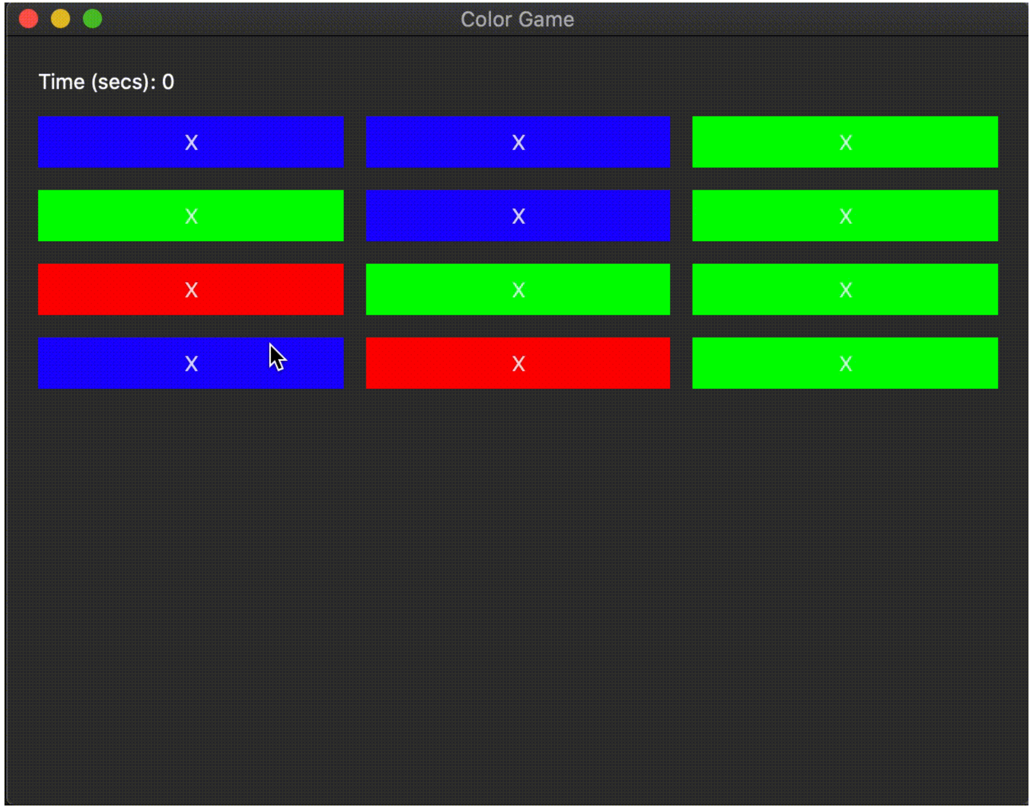
Example gif for winning scenario (it is speeded-up)



2. Color Game

Finally, the design looks like this:

Example gif for losing scenario (it is speeded-up)



QT on terminal

- Existing project

```
qmake -makefile
make
```

- New project

```
qmake -project // create an qmake project file
qmake //  create a "Makefile" which contains the rules to build your application
make
```

Conclusion

We have seen QT Creator and several QT widgets with examples.

Any questions?